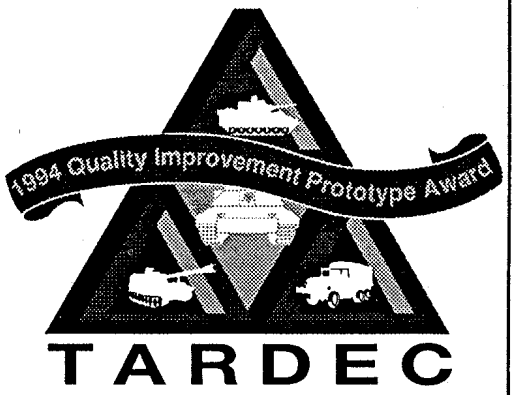
TARDEC

---TECHNICAL REPORT---

THE NATION'S LABORATORY FOR ADVANCED AUTOMOTIVE TECHNOLOGY

WINNER OF THE 1994 FEDERAL QUALITY IMPROVEMENT PROTOTYPE AWARD

No. 13825



DYNAMIC LINK LIBRARY OF INTERPOLATION/TABLE LOOKUP FUNCTIONS FOR 20SIM

By Wesley Bylsma

U.S. Army Tank-Automotive Research,
Development, and Engineering Center
Detroit Arsenal
Warren, Michigan 48397-5000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	SEPTEMBER 2002	JULY – SEPTEMBER 2002

4. TITLE AND SUBTITLE
DYNAMIC LINK LIBRARY OF INTERPOLATION/TABLE LOOKUP FUNCTIONS FOR 20SIM

5. FUNDING NUMBERS

6. AUTHOR(S)
Wesley Bylsma

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Tank-automotive and
Armaments Command/National
Automotive Center
ATTN: AMSTA-TR-N/MS157
Warren, MI 48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
13825

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release: Distribution is unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 Words)*

A dynamic link library (DLL) is developed to perform interpolation/table lookup functions from within 20-SIM by Controllab Products B.V. (Netherlands). The use of these functions, implemented in the C programming language, provides a multi-platform modeling and simulation environment through 20-SIM's C code generation capability. Externally developed interpolation/table lookup functions are necessary since 20-SIM's internal table lookup functions are not allowed with its C code generation process.

14. SUBJECT TERMS
20-SIM, dynamic link library, interpolation, table lookup

15. NUMBER OF PAGES
19

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

Table of Contents

Technical Report 13825
September 2002

DYNAMIC LINK LIBRARY OF INTERPOLATION/TABLE LOOKUP FUNCTIONS FOR 20SIM

Wesley Bylsma
U.S. Army Tank-automotive and Armaments Command
Research, Development and Engineering Center
National Automotive Center
Warren, Michigan 48397-5000

ABSTRACT
A dynamic link library (DLL) is developed to perform interpolation/table lookup functions from within 20-SIM by Controllab Products B.V. (Netherlands). The use of these functions, implemented in the C programming language, provides a multi-platform modeling and simulation environment through 20-SIM's C code generation capability. Externally developed interpolation/table lookup functions are necessary since 20-SIM's internal table lookup functions are not allowed with its C code generation process.

1.0 INTRODUCTION

The modeling and simulation software 20-SIM (www.20sim.com) has been developed at the Control Laboratory of the University of Twente (the Netherlands). It fully supports graphical and rapid modeling through iconic diagrams, block diagrams, bond graphs and equations in an unlimited hierarchical model structure--allowing the design and analysis of dynamic systems in an intuitive and user friendly way. While 20-SIM is based mainly on the Microsoft Windows PC platform, it provides a tool for C code generation. This capability provides a means by which a model can be run on another platform as long as that platform supports the C programming language. Some of 20-SIM's built-in functions, such as table lookups, are not supported by the code generation process. The use of calls to dynamic link libraries (DLL's) is supported and these calls are included in the C code generated. By writing needed interpolation/table lookup functions in C, these functions can be used on any platform—by calls to a DLL (Windows) or calls directly to a C function compiled and linked with the model source code. To enable this multi-platform capability, a dynamic link library of interpolation/table lookup functions has been written in C.

2.0 DESCRIPTION

While 20-SIM does not allow some of its built-in lookup functions to be used in the C code generation process it does allow calls to DLL's (subroutines). Two types of data need to be handled through this process: Internal arrays and external arrays. Internal arrays are declared directly within 20-SIM. External arrays are usually large sets of data that are stored in files outside 20-SIM (such as road profiles, etc.). They are read in during the initialization process and kept in temporary memory for access during the simulation. This requires some internal static storage space within the DLL to hold the external arrays values. The internal and external arrays described here are limited to two dimensions.

2.1 20-SIM Model

In 20-SIM every model or submodel has an implementation. This can be a composition of lower level submodels, which themselves are composed of lower level submodels etc. At the bottom of this hierarchy the submodels consist of a set of mathematical equations (equation submodel). These submodels are therefore known as equation models. All equations used in 20-SIM are described in the language SIDOPS+.

The general layout of a SIDOPS+ equation model is:

```
constants
        //enter your constants here
parameters
        //enter your parameters here
variables
        //enter your variables here
initialequations
        // enter your initial equations here
code
        // enter your equations here
equations
        // enter your equations here
finalequations
        // enter your final equations here
```

Only the "equations" section is required. The other sections are optional. It is within this framework that the functions will be called. A call to a DLL function in 20-SIM is defined as

```
out = dll('dll_func.dll','func_name',input);
```

and could appear in any of the sections that allow equations. This function call will be used to replace calls to the "data" and "table" functions.

2.2 DLL Function Prototype

The C function prototype required for DLL functions is

```
int func_name(double *inarr, int inputs,
    double *outarr, int outputs, int major) ;
```

where

- `inarr`: pointer to an input array of doubles. The size of this array is given by the second argument.
- `inputs`: size of the input array of doubles.
- `outarr`: pointer to an output array of doubles. The size of this array is given by the fourth argument.
- `outputs`: size of the output array of doubles.
- `major`: boolean which is 1 if the integration method is performing a major integration step, and 0 in the other cases. For example Runge-Kutta 4 method only has one in four model evaluations a major step.

This call only passes a pointer and length to the input and output variables (and a flag to indicate a minor or major integration step) which does not pass enough information to call the interpolation/table lookup routine at one time. Because of this calling restriction, multiple DLL functions must be defined to set the necessary parameters before the interpolation/table lookup function call. This requires some internal static storage space (global) within the DLL to store these parameters (initial values).

2.3 DLL Internal Storage

Although this is not needed for external arrays--since the complete array information is contained within the static array structure when it is read in and allocated into static memory (see `AllocateArray` in section 2.4)—the same mechanism is used to avoid more than one interpolation/table look up function for both types of arrays. Therefore, for both external and internal arrays, multiple function calls are made to initialize these parameters before the interpolation/table lookup function call.

The necessary internal static storage needed to access external array information during the simulation and to accommodate this parameter initialization is defined by the following global structure/variables at the beginning of the DLL.

```
#define FLEN 512  // filename length
#define NARR 10  // max # of arrays
#define ALEN 30000  // max array length

typedef struct {
  char filename[FLEN];
  double mem[ALEN];
  int size;
  int dimy;
  int dimx;
  } arrstruct;  // external array struct

// external array structure
static arrstruct arr[NARR];  // external arrays
static int arrnum;  // external array # to work on

// interpolation/table lookup parameters space
static double *arrx;  // pointer to x array
static double *arry;  // pointer to y array
static double *arrxy;  // pointer to xy array
static int size;  // dimension of array (1 or 2)
static int dimy;  // length of y dimension
static int dimx;  // length of x dimension
```

This structure is defined so that pointers to the arrays (external or internal), array size, and array dimension are set before the interpolation/table lookup function is called. A description for the set of function calls that implement this multi-call scheme is given first for external arrays (section 2.4) and then internal arrays (section 2.5).

2.4 External Array Functions

The external array functions are described below and each operates on the global structure. All have the same prototype for DLL's as described in section 2.2:

`SetArrNum`

Sets "`arrnum`" to the number in the first location of the input pointer passed (`inarr`). Defines what array in "`arr[]`" to work on.

`SetFileName`

Copies the string pointed to by the input pointer into "`arr[arrnum].filename`" Prerequisite: Must call `SetArrNum` to define array number to work on.

`AllocateArray`

Reads into "`arr[arrnum].mem`" the data contained in "`arr[arrnum].filename`" and sets "`arr[arrnum].size`", "`arr[arrnum].dimy`", and "`arr[arrnum].dimx`". Prerequisite: Must call `SetArrNum` to define array number to work on.

The data file must be in the following format ("%" denotes comments and "%%" denotes a command—both can appear only in the header):

```
---BOF---
%
% %%NDIM:# defines the number of
% dimensions (1 or 2)
%
% %%DIMS:# defines the length of each
% dimension(or %%DIMS:# # if 2 dimension)
%
% file name
%%NDIM:1
%%DIMS:10
0.000000
3.000000
.
.
.
---EOF---
```

SetXdll
Copies the pointer to "arr[arrnum].mem" to the x array pointer "arrx". Prerequisite: Must call SetArrNum to define array number to work on.

SetYdll
Copies the pointer to "arr[arrnum].mem" to the y array pointer "arry". Prerequisite: Must call SetArrNum to define array number to work on.

SetXYdll
Copies the pointer to "arr[arrnum].mem" to the xy array pointer "arrxy". Prerequisite: Must call SetArrNum to define array number to work on.

InterpDll
Uses the pointers in "arrx", "arry", "arraxy", "arr[arrnum].size", "arr[arrnum].dimy", and "arr[arrnum].dimx" to perform an interpolation using the first location of the input pointer as lookup value "x" and the second location (if size = 2) of the input pointer as lookup value "y". The value obtained is passed back to the first location pointed to by the output pointer (outarr). Prerequisite: Must call SetArrNum to define array number to work on. This function can be replaced with a call to "SetSize" and "Interp".

2.5 Internal Array Functions

The internal array functions are described below and each operates on the global structure. All have the same prototype for DLL's as described in section 2.2.

SetSize
Sets "size", "dimy", and "dimx" to the number in the first, second, and third location of the input pointer passed, respectively.

SetX
Copies the input array pointer to the x array pointer "arrx".

SetY
Copies the input array pointer to the y array pointer "arry".

SetXY
Copies the input array pointer to the xy array pointer "arrxy".

Interp
Uses the pointers in "arrx", "arry", "arraxy", "size", "dimy", and "dimx" to perform an interpolation using the first location of the input pointer as lookup value "x" and the second location (if size = 2) of the input pointer as lookup value "y". The value obtained is passed back to the first location pointed to by the output pointer (outarr).

2.6 DLL Support Functions

Several other miscellaneous functions are described below:

Initialize
Used to perform necessary initialization tasks when the DLL is loaded.

Terminate
Used to perform necessary termination tasks when the DLL is unloaded.

printnum
Searches through "in1[]" for each name in "list1[]" and prints out the array index of the location where the name is found. This function is used to identify the array location number for parameters, variables, etc. for direct modification of values in the generated C code. The prototype of this function is

```
int printnum(char *in1[], int in1, char *list1[],
             int list1)
```

where

in1[]: list of names to search for.

in1: size of "in1[]" input array of names.

list1[]: list of defined names to search in.

list1: size of "list1[]" search array of names.

3.0 FUNCTION IMPLEMENTATION

During the actual implementation of these functions issues arise that should be noted. These are platform specific. This particular implementation was accomplished using Borland C++ Builder 6 for testing the DLL with 20-SIM on the PC platform, and then code was transferred to an SGI Irix (Unix) platform.

3.1 Creating the DLL Functions

For the PC platform (using Borland C++ Builder 6), the "`extern "C"`" and "`__declspec(dllexport)`" specifiers are required to export functions, data, and objects from a DLL. An example of this for the `SetArrNum` function is

```
extern "C" __declspec(dllexport)int SetArrNum(double
*inarr, int inputs, double *outarr, int outputs, int
major) ;
```

See Appendix A for the particular DLL entry point subroutine. (Specific to the Borland C++ Builder is the inclusion of the "`.LIB`" file in the project which is made with the `IMPLIB` command.)

3.2 Modifying the 20-SIM Function Calls

The following illustrates the equation model sections and calling sequence needed to handle **external** arrays. (Notice the underscore ("_") prefix to the function name due to Borland C++ Builder)

```
parameters
// string filename;
// integer column;
   string fncyclex = 'fmtv_cycle_x.arr';
   string fncycley = 'fmtv_cycle_y.arr';

variables
   real parm[3];
   real tmp;

initialequations
if TRUE then
   parm[1] = 1.0;
   tmp = dll('dllinterp.dll','_SetArrNum',parm);
   tmp = dll('dllinterp.dll','_SetFileName',
         fncyclex);
   tmp = dll('dllinterp.dll','_AllocateArray',parm);
   parm[1] = 2.0;
   tmp = dll('dllinterp.dll','_SetArrNum',parm);
   tmp = dll('dllinterp.dll','_SetFileName',
         fncycley);
   tmp = dll('dllinterp.dll','_AllocateArray',parm);
end;

equations
if TRUE then
   parm[1] = 1.0;  // array index
   tmp = dll('dllinterp.dll','_SetArrNum',parm);
   tmp = dll('dllinterp.dll','_SetXdll',parm);
   parm[1] = 2.0;  // array index
   tmp = dll('dllinterp.dll','_SetArrNum',parm);
   tmp = dll('dllinterp.dll','_SetYdll',parm);
   parm[1] = time;  // lookup value
   output  = dll('dllinterp.dll','_InterpDll',parm);
end;

//20SIM function replaced
// output = data (filename, column);
```

A summary of the process is as follows—a string is declared for each array file to be used. File "`fmtv_cycle_x.arr`" is read into array number 1 and file "`fmtv_cycle_y.arr`" is read into array number 2. (array indexes can start at zero since arrays are zero based in C). Storage of each array is done during initialization (`initialequations` section) so that the data is available during the simulation. The array number and filename are set before they are read in using "`AllocateArray`". During the simulation, the array number is set to point to the appropriate array (X or Y) and then a pointer to it is copied into the global static parameters used by the interpolation function. The size of the array indexed by "`SetArrNum`" determines how many lookup indexes to pass (one, `time` in this case). Notice the inclusion of the "`if TRUE then`" and "`end;`" bracketing of the calling sequence. This is necessary to prevent 20-SIM from trying to reorder the equations. The "`code`" section can also be used for this purpose. String variables can also be used in the DLL call for the DLL name and the DLL function name.

The following illustrates the equation model sections and calling sequence needed to handle **internal** arrays.

```
parameters
   real engdat_x[12];
   real engdat_y[22];
   real  engdat[22,12];

variables
   real parm[3];
   real tmp;

equations
if TRUE then
   tmp = dll('dllinterp.dll','_SetX',engdat_x);
   tmp = dll('dllinterp.dll','_SetY',engdat_y);
   tmp = dll('dllinterp.dll','_SetXY',engdat);
   parm[1] = 2.0;  // array dimensions
   parm[2] = 22;  // size of y dimension
   parm[3] = 12;  // size of x dimension
   tmp = dll('dllinterp.dll','_SetSize',parm);
   parm[1] = fuel;  // x lookup value
   parm[2] = omega;  // y lookup value
   z = dll('dllinterp.dll','_Interp',parm);
end;
```

A summary of the process is as follows--during the simulation, a pointer to each array is copied into the global static parameters used by the interpolation function. Notice that two lookup indexes were passed (`fuel` and `omega`) in this case since the size of the array was specified to be two (see "`SetSize`").

3.3 Generating the C Code

The code generation process within 20-SIM should be capable of producing C code for the model or submodel once the 20-SIM functions have been replaced with the DLL calls. The files generated are named with a prefix of "`xx`" (eg. "`xxmodel.c`"). Modification of the "`targets.ini`" file allows the prefix "`xx`" to be changed to another desired prefix string.

3.4 Renaming/Recompiling the DLL functions

The calls to the DLL functions from the generated C code will have the following structure

```
dllinterp_Initialize ();
dllinterp__myGetFile ("eng_trq.dat", 1, &xx_V[263],
                      1, xx_major);
```

In the C code generation process, the name of the DLL being called, in this case "`dllinterp`", has been added to the beginning of the DLL function name. This prefix will have to be added to the DLL function/subroutine names before compiling on another platform. It should be noted, as before, that some function names may also require a leading underscore ("_") __in the 20SIM environment. This is the case for the Borland C++ Builder since it automatically appends an underscore as a prefix to the function names. The use of "`extern "C"`" may also be required to avoid mangled function names in the C++ environment.

Once the DLL functions (used on the 20-SIM platform) have been renamed with the DLL name as a prefix, they can be recompiled on another platform and linked into the model executable. The DLL functions will now be called directly by the model executable.

3.5 FORTRAN Interface

Since the model/submodel from 20-SIM will be generated directly into C code, calling it from another C program should be trivial. Calling the model from FORTRAN presents a set of interface issues that will be discussed, but are implementation specific. For the particular version (MIPSProF77) and platform SGI Irix used here, calling C functions from FORTRAN requires the C functions to be declared "`EXTERNAL`", "`double`" C variables be declared "`DOUBLE PRECISION`", and "`int`" C variables to be declared "`INTEGER`"—such as

```
      DOUBLE PRECISION u(5),y(5)
      INTEGER maj
      EXTERNAL XXInitializeSubmodel
      EXTERNAL XXCalculateSubmodel
      EXTERNAL XXTerminateSubmodel
```

The model call looks like

```
      CALL  XXCalculateSubmodel(%ref(u), %ref(y),
     &                          %val(time))
```

where "`%ref()`" passes the variable by reference and "`%val`" passes the variable by value. The variable "`time`" is passed by value (see the function prototype in "`XXCalculateSubmodel`"). If your implementation of FORTRAN does not allow passing by value, the model/submodel subroutines and their prototype will have to be modified to accept a reference to the variable and then use a temporary variable to get the value from the reference passed to the function.

Modification can also be made to the "XXCalculateSubmodel" subroutine to pass back more variables through modification of the calling prototype. Finally, the model functions (or any C function called)

```
XXInitializeSubmodel
XXCalculateSubmodel
XXTerminateSubmodel
```

should be renamed as lowercase with an "_" as a suffix to satisfy the FORTRAN calling interface to C functions.

3.6 Interface Issues

A mechanism must be implemented to ensure that the initialization "`XXInitializeSubmodel`" and termination "`XXTerminateSubmodel`" functions are called only once, at the beginning and end of the simulation, respectively.

The code generation process does include simple integrators (eg. Euler, Runge-Kutta (RK2)) as part of the model code. Consideration must be given as to how the integration of state variables will be handled. One method is to use the integrators included with the model and make sure its time steps are synced with the calling program's, or the second method is to have the calling program's integrator handle the state variables. The second method involves replacing calls to the built-in integrator with a process that will have the calling program update the state variables directly. This issue arises when calling "`XXCalculateSubmodel`"

While 20-SIM defines the function prototype required for DLL functions, it does not do type checking on them. The passing of a string to the "`SetFileName`" function in 20-SIM is permissible through the use of the "union" structure

```
char *NamePtr;
union
{
  double realval;
  char *strval;
} value;
value.realval = inarr[0];
NamePtr = value.strval;
```

However, when the C code is generated it will place a string variable (type "`*char`") as the first argument to the "`SetFileName`" function (as the input array). Compiling the code will result in a type mismatch. Outside of the 20-SIM environment, we are no longer restricted to use the DLL required prototype and should thus change the first argument of the prototype to this function to be "`*char`".

3.7 Support Function Usage

The "`printnum`" function is used to identify the array location number for parameters, variables, etc. for direct modification of values in the generated C code.

Currently in 20-SIM, there are six arrays in which to search for names:

1. xx_constant_names[]
2. xx_parameter_names[]
3. xx_variable_names[]
4. xx_state_names[]
5. xx_rate_names[]
6. xx_matrix_names[]

A call to "`printnum`" for each one may be in order if you are not sure which array the name you are looking for is in. The file "`xxmodel.c`" defines the dimension of these name arrays. For each name array there is a corresponding value array:

1. xx_constants[]
2. xx_parameters[]
3. xx_variables[]
4. xx_states[]
5. xx_rates[]
6. xx_matrices[]

and a pointer to each of these value arrays

1. xx_C
2. xx_P
3. xx_V
4. xx_S
5. xx_R
6. xx_M

An example of using the "`printnum`" function is given below. The following code should be inserted into the generated C code "`xxmain.c`" to find the named variables/state array indexes.

```
char *list[]={"cycle\\output",
      "Submodel6\\Accumulator\\Function\\output",
      "VehiclePitch\\tire_rear2\\X\\output",
      "Driver\\ms_mph\\output",
      "Submodel6\\Engine\\omega_rpm\\output",NULL};

printf("xx_variable_names\n");
printnum(&xx_variable_names[0],3033,&list[0],5);
printf("xx_state_names\n");
printnum(&xx_state_names[0],41,&list[0],5);
exit(0);
```

A summary of the process is as follows--here "`*list[]`" is an array of names you want to find an index for with length five. Two calls are made, one to search the variable name array, with length 3033, and one to search the state name array, with length 41. The output below indicates what index value you should look for in the generated C code ("`xxmodel.c`") to change or modify the named value. (Remember to remove the "`exit(0)`" command and comment out this section when finished.) Sample results of calling this function are below

```
xx_variable_names
Total:5
0. cycle\output [1772]
1. Submodel6\Accumulator\Function\output [2769]
3. Driver\ms_mph\output [1800]
4. Submodel6\Engine\omega_rpm\output [2172]

xx_state_names
Total:5
2. VehiclePitch\tire_rear2\X\output [18]
```

To find the "`Driver\ms_mph\output`" variable in the generated C code for the model you would look for "`xx_variables[1800]`" or "`xx_V[1800]`".

4.0 SUMMARY/CONCLUSION

A dynamic link library (DLL) is developed to perform interpolation/table lookup functions from within 20-SIM. The use of these functions allows 20-SIM models to be ported to other platforms through the use of its C code generation capability. A set of DLL functions (C subroutines) were developed to handle external and internal arrays with a maximum dimension of two.

CONTACT

The author is a an engineer at the U.S. Army Tank-automotive and Armaments Command, Research, Development and Engineering Center (TACOM-TARDEC). Interested parties can contact the author at the U.S. Army Tank-automotive and Armaments Command, ATTN: AMSTA-TR-N/MS157, Warren, Michigan 48397-5000, bylsmaw@tacom.army.mil.

REFERENCES

"Getting Started with 20-Sim" (Version 3.2), Controllab Products B.V., Enschede, Netherlands. December 2001.

DEFINITIONS, ACRONYMS, ABBREVIATIONS

DLL – Dynamic Link Library
TACOM - U.S. Army Tank-automotive and Armaments Command
TARDEC - TACOM Research, Development and Engineering Center
NAC - National Automotive Center

APPENDIX A – DLL Listing (Borland C++ Builder 6)

```
//---------------------------------------------------------------------------
#include <string.h> //strcmp
#include <stdio.h>
#include <stdlib.h>  /* calloc */
#include <string.h>  /* strcpy */
#include <windows.h>
//---------------------------------------------------------------------------
//   Important note about DLL memory management when your DLL uses the
//   static version of the RunTime Library:
//
//   If your DLL exports any functions that pass String objects (or structs/
//   classes containing nested Strings) as parameter or function results,
//   you will need to add the library MEMMGR.LIB to both the DLL project and
//   any other projects that use the DLL.  You will also need to use MEMMGR.LIB
//   if any other projects which use the DLL will be performing new or delete
//   operations on any non-TObject-derived classes which are exported from the
//   DLL. Adding MEMMGR.LIB to your project will change the DLL and its calling
//   EXE's to use the BORLNDMM.DLL as their memory manager.  In these cases,
//   the file BORLNDMM.DLL should be deployed along with your DLL.
//
//   To avoid using BORLNDMM.DLL, pass string information using "char *" or
//   ShortString parameters.
//
//   If your DLL uses the dynamic version of the RTL, you do not need to
//   explicitly add MEMMGR.LIB as this will be done implicitly for you
//---------------------------------------------------------------------------
#define FLEN 512
#define NARR 10  // 10 arrays
#define ALEN 30000
typedef struct {
  char filename[FLEN];
  double mem[ALEN];
  int size;
  int dimy;
  int dimx;
  } arrstruct;

static arrstruct arr[NARR];
```

```
static int arrnum;
static double *arrx;
static double *arry;
static double *arrxy;
static int size;
static int dimy;
static int dimx;

int interp1(double *x, double *y, int l, double xi, double *yo);
int interp2(double *x, double *y, double *z, int m, int n, double xi, double yi, double *zo);
int readtbl2d(const char *file, double *arr2d);
extern "C" __declspec(dllexport)int SetArrNum(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetFileName(double *inarr, int inputs, double *outarr, int outputs, int major) ;
//extern "C" __declspec(dllexport)int SetFileName(char *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int AllocateArray(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetX(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetXdll(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetY(double *inarr, int inputs, double *outarr, int outputs, int major) ;

extern "C" __declspec(dllexport)int SetYdll(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetXY(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetSize(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int SetXYdll(double *inarr, int inputs, double *outarr, int outputs, int major) ;

extern "C" __declspec(dllexport)int Interp(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int InterpDll(double *inarr, int inputs, double *outarr, int outputs, int major) ;
extern "C" __declspec(dllexport)int Initialize();
extern "C" __declspec(dllexport)int Terminate();
extern "C" __declspec(dllexport)int printnum(char *in1[], int inl, char *list1[], int listl);

#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        return 1;
}
//---------------------------------------------------------------------------
```

```
int Initialize()
{
  return 0;  // do nothing now
}
```

```
int Terminate()
{
  return 0;  // do nothing now
}
```

```
/* printnum   */
int printnum(char *in1[], int inl, char *list1[], int listl)
{
  int i,j,k;
  printf("Total:%d\n",listl);
  for (j=0;j<listl;j++)
  {
    for (i=0;i<inl;i++)
    {
      k = strcmp(list1[j],in1[i]);
      if (k==0)
      {
        printf("%d. %s [%d]\n",j,in1[i],i);
        break;
      }
    }
  }
  return 0;
}
```

```
int SetArrNum(double *inarr, int inputs, double *outarr, int outputs, int major)
{
      arrnum = (int) *inarr;
      return 0;
}
```

```
int SetFileName(double *inarr, int inputs, double *outarr, int outputs, int major)
//int SetFileName(char *inarr, int inputs, double *outarr, int outputs, int major)
{
  char *NamePtr;
    union
  {
    double realval;
    char *strval;
  } value;
  value.realval = inarr[0];
  NamePtr = value.strval;

      strcpy(arr[arrnum].filename,NamePtr);
      return 0;
}
```

```
#define BUFLEN 1024
int AllocateArray(double *inarr, int inputs, double *outarr, int outputs, int major)
{
/* readtbl2d.c
   Reads in 2d table from file according to format.
   02-03-27 Update syntax.
   02-03-18 Created.
   ---INPUT FILE FORMAT---
   %%NDIM:# dimenstions
   %%DIMS:dim1 dim2 dim3 dim4 ...
   val()
   %comment
   val()
   ...
*/

  /* ---VARIABLES--- */

  FILE *fin;  /* file handle */
  int val;  /* scanned value */
  int ndim;  /* number of dimensions */
```

```
int dims[2];  /* size of dimensions read in */
char buff[BUFLEN];  /* read buffer */
int row;  /* row counter */
int col;  /* col counter */
double dtmp;  /* temporary double */
int i,j;
fin = fopen(arr[arrnum].filename,"rt");
fseek(fin,0,SEEK_SET);
if (fin == NULL)
{
  fclose(fin);
  return -1;
}
/* ---PROCESS HEADER--- */
ndim = dims[0] = dims[1] = row = col = 0;
while (fgets(buff,BUFLEN,fin) != NULL)
{
  if (strncmp(buff,"%%",2)==0)
  {
    if (strncmp(&buff[2],"NDIM:",5)==0)
    {
      val = sscanf(&buff[7],"%d\n",&ndim);
      arr[arrnum].size = ndim;
    }
    else if (strncmp(&buff[2],"DIMS:",5)==0)
    {
      if (ndim == 2)
      {
        val = sscanf(&buff[7],"%d %d\n",&dims[0],&dims[1]);
        arr[arrnum].dimy = dims[0];
        arr[arrnum].dimx = dims[1];
        if (dims[0] > ALEN || dims[1] > ALEN)
        {
            fclose(fin);
          return -1;
        }
      }
      if (ndim == 1)
      {
```

```
          /*keep dim format as if for 2d - col in dim[1] */
          val = sscanf(&buff[7],"%d\n",&dims[1]);
          dims[0]= 1;
          arr[arrnum].dimy = dims[0];
          arr[arrnum].dimx = dims[1];
        }
      }
    }
    else if(strncmp(buff,"%",1)!=0)
    {
      break;
    }
  }
  if ((ndim != 1) && (ndim != 2))
  {
    fclose(fin);
    return -1;
  }
   val = sscanf(&buff[0],"%lf\n",&dtmp);

  *(arr[arrnum].mem+row*dims[1]+col)=dtmp;
  col++;
/* check col bound here */
/*  if ((ndim != dim[0]) || (dims[0] != dim[1]) || (dims[1] != dim[2]))
  {
    exit(0);
    return -1;
  } */

  /* ---PROCESS DATA--- */
  while (fgets(buff,BUFLEN,fin) != NULL)
  {
    if (strncmp(buff,"%",1)==0) continue;

    val = sscanf(&buff[0],"%lf\n",&dtmp);

    *(arr[arrnum].mem+row*dims[1]+col)=dtmp;
    col++;
    if (col >= dims[1])
```

```
    {

      row++;
      col = 0;
      /* check row bound here */
    }
//    if (row >= dim[1]) break;
    if (ndim != 1) {
      if (row >= dims[0]) break;
    }
  }
  fclose(fin);

  return 0;
}
```

```
int SetX(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  arrx = inarr;
  return 0;
}
```

```
int SetXdll(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  arrx = arr[arrnum].mem;
  return 0;
}
```

```
int SetY(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  arry = inarr;
  return 0;
}
```

```
int SetYdll(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  arry = arr[arrnum].mem;
  return 0;
}
```

```
int SetXY(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  arrxy = inarr;
  return 0;
}
```

```
int SetSize(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  size = (int) *(inarr);
  dimy = (int) *(inarr+1);
  dimx = (int) *(inarr+2);
  return 0;
}
```

```
int SetXYdll(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  arrxy = arr[arrnum].mem;
  return 0;
}
```

```
/* must set arrx,arry,arrxy before call */
int InterpDll(double *inarr, int inputs, double *outarr, int outputs, int major)
{
   int ierr;
  double xd,yd,tmp;

  xd = *(inarr);
  yd = *(inarr+1);
  if (arr[arrnum].size == 1)
  {
    ierr = interp1(arrx,arry,arr[arrnum].dimx, xd, outarr);
  }
  else
  {
    ierr = interp2(arrx,arry,arrxy,arr[arrnum].dimx,arr[arrnum].dimy, xd,yd, outarr);
  }
  return 0; //return successful
}
```

```
int Interp(double *inarr, int inputs, double *outarr, int outputs, int major)
{
  int ierr;
  double xd,yd;

  xd = *(inarr);
  yd = *(inarr+1);
  if (size == 1)
  {
    ierr = interp1(arrx,arry,dimx, xd, outarr);
  }
  else
  {
    ierr = interp2(arrx,arry,arrxy,dimx,dimy, xd,yd, outarr);
  }
  return 0; //return successful
}
```

```
/* interp1.c
   This is a 1D interpolation subroutine.  Given x,y and xi return
   interplated value yo.  Return function value is index s or -s if
   out of range.  If out of range, yo is extrapolated.

   02-03-27 Updated syntax.
   02-03-11 Converted to C from F90.  Indexes in C are zero "[0]" based.
            Adjust s, e, pvt's accordingly.
*/

//#include <stdio.h>
int interp1(double *x, double *y, int l, double xi, double *yo)
{
  /* ---VARIABLES--- */
  int s;  /* start index */
  int e;  /* end index */
  int pvt1;  /* pivot pt 1 */
  int pvt2;  /* pivot pt 2 for even */
  int i;  /* process counter */
```

```
s = 0;  /* start at very beginning */
e = l - 1;  /* start at very end */

/* check for range error */
if (xi < *(x))
{
  /* if no extrapolation  use -> *yo = *(y); */
  /* extrapolate */
  s = 0;
  e = s + 1;
 // *yo = *(y + s) + (xi - *(x + s)) * ( *(y + e) - *(y + s) ) / ( *(x + e) - *(x + s) );
  *yo = *(y);
//  return -s;  /* (-) means out of range */
  return s;  /* no extrap*/
}
else if ( xi > *(x + l - 1))
{
  /* if no extrapolation use -> *yo = *(y + l - 1); */
  /* extrapolate */
  e = l - 1;
  s = e - 1;
//  *yo = *(y + s) + (xi - *(x + s)) * ( *(y + e) - *(y + s) ) / ( *(x + e) - *(x + s) );
  *yo = *(y + l - 1);
//  return -s; /* (-) means out of range */
  return s; /* no extrap */
}

/* ---PROCESS LOOP--- */
for (i = 0; i < l; i++)
{
  if ((e - s) % 2 == 0 )
  {
    /* ---EVEN--- */
    pvt1 = s + (e - s) / 2;
    if (xi == *(x + pvt1))
    {
      *yo = *(y + pvt1);
      return pvt1;
    }
```

```
      if (xi > *(x + pvt1))
      {
        s = pvt1;  /* top half */
      }
      else
      {
        e = pvt1;  /* bottom half */
      }
    }
    else
    {
      /* ---ODD--- */
      pvt1 = s + (e - s - 1) / 2;
      pvt2 = pvt1 + 1;
      if (xi > *(x + pvt2))
      {
        s = pvt2;  /* top half */
      }
      else if (xi < *(x + pvt1))
      {
        e = pvt1;  /* bottom half */
      }
      else
      {
        s = pvt1;  /* between these */
        e = pvt2;
      }
    }
    if ((e - s) <= 1)
    {
      /* ---FINAL ANSWER--- */
      *yo = *(y + s) + (xi - *(x + s)) * ( *(y + e) - *(y + s) ) / ( *(x + e) - *(x + s) );
      return s;
    }
  }
  return -1;
}
```

```
/* interp2.c
   This is a 2D interpolation subroutine.  Given x,y,z,xi and yi return
   interplated value zo.  Return function value is -s if
   out of range. Uses interp1

   02-03-27 Updated syntax.
   02-03-11 Converted to C from F90.  Indexes in C are zero "[0]" based.
*/

//#include <stdio.h>
/*int interp1(double *x, double *y, int l, double xi, double *yo)*/

int interp2(double *x, double *y, double *z, int m, int n, double xi, double yi, double *zo)
{
  /* ---VARIABLES--- */
  int sx;  /* start x index */
  int sy;  /* start y index */
  int ierr;  /* error return code */
  double valy;  /* interpolated value of y1 */
  double valy1;  /* interpolated value of y2 */
  double tmp;  /* temporary variable */

  /* ---GET INDEXES--- */
  sx = interp1(x,x,m,xi,&tmp);
  sy = interp1(y,y,n,yi,&tmp);

  /* ---IF YOU DO NOT WANT EXTRAPOLATION UNCOMMENT THIS CODE---
  % if (sx==-1 || sy==-1)
  % {
  %   *(zo) = -1;
  %   return -1;
  %  }
  */

  /* ---GET Y'S FROM X'S--- */


  /* take care of divide by zero case */
```

```
  if ( *(x + (sx+1)) == *(x + sx) )
  {
    valy = *(z + sy*m + sx);
    valy1 = *(z + (sy+1)*m + sx);
  }
  else
  {
    valy = *(z + sy*m + sx) + (xi - *(x + sx)) * ( *(z + sy*m + (sx+1)) - *(z + sy*m + sx) ) / ( *(x + (sx+1)) - *(x + sx) );
    valy1 = *(z + (sy+1)*m + sx) + (xi - *(x + sx)) * ( *(z + (sy+1)*m + (sx+1)) - *(z + (sy+1)*m + sx) ) / ( *(x + (sx+1)) - *(x + sx) );
  }
  /* ---GET Z FROM Y'S--- */

  if ( *(y + (sy+1)) == *(y + sy) )
  {
    *zo  = valy;
  }
  else
  {
    *zo  = valy + (yi - *(y + sy)) * (valy1 - valy) / ( *(y + (sy+1)) - *(y + sy) );
  }
  tmp = *zo;
  return 0;
}
```